PLAN

Présentation

JUnit

Exemple

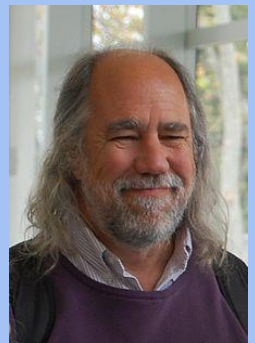
Conclusion

Les tests d'intégration continue

Louis Le Brun
Cyrille Michard
Romain Pasche

L’intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.

Grady Booch



-Limiter les problèmes d'intégration lors du développement afin de réduire les risques de bug.

-Automatiser l'exécution des suites de tests.

-Voir l'évolution du développement du logiciel.

Jenkins

Hudson

JUnit

- Fichier junit.jar → répertoire où se trouvent les classes et les packages Java afin qu'elle les exécute. Hamcrest.jar doit lui aussi être importé depuis une mise à jour récente.

- Dans le fichier test du projet il faut importer les packages org.junit pour pouvoir faire les test,

org.junit.After org.junit.AfterClass org.junit.Beforeorg.junit.BeforeClass

- Les cas de tests utilisent différentes méthodes proposées par le framework.
- Elles commencent toutes par “assert”

    assertEquals()

    assertFalse()

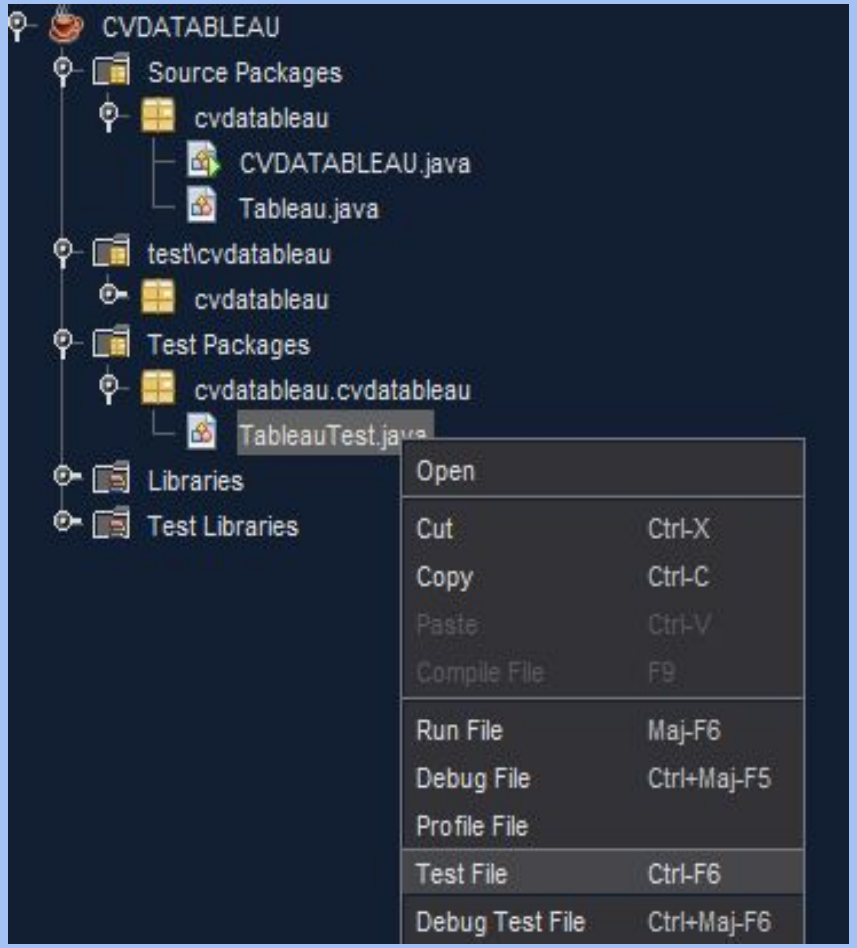
    assertNull()

    ...

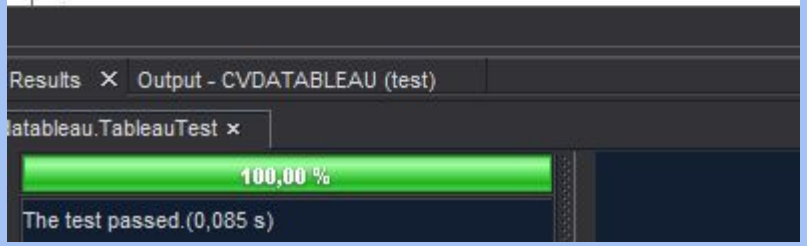
- Fichiers “Class” différents avec les différents tests
- Les tests s’effectuent à des moments différents (@)
- assertEquals(4+5,9)

```java
package cvdatableau;

public class Tableau {

    private int tableau[];
    private int longueur;

    public Tableau(int longueur) {
        this.longueur = longueur;
        this.tableau = new int[longueur];

    }

    public Tableau(){
        this.longueur = 5;
        this.tableau = new int[5];
    }

    public static void remplirTab(int[] tableau){
        for(int i = 0 ; i < tableau.length ; i++){
            tableau[i]=0;
        }
    }

    public static void supprOcc(int[] tableau, int element){
        for(int i = 0; i < tableau.length; i++){
            if(tableau[i] == element) tableau[i] = -1;
        }
    }



}
```

```java
public class TableauTest {

    public TableauTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    @org.junit.Test
    public void testSomeMethod() {
        int tab[] = new int[5];

        Tableau.remplirTab(tab);
        for(int i = 0; i< tab.length ; i++){
            assertEquals(tab[i],0);
        }

        Tableau.supprOcc(tab,0);
        for(int i = 0; i< tab.length ; i++){
            assertEquals(tab[i],-1);

        }

    }
```

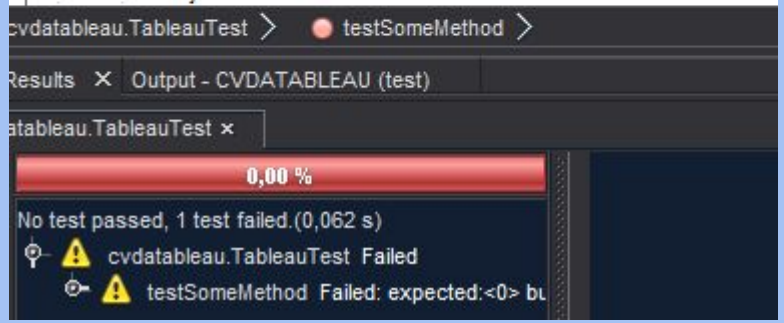


```
Tableau.remplirTab(tab);
for(int i = 0; i< tab.length ; i++){
    assertEquals(tab[i],0);
}

Tableau.supprOcc(tab,0);
for(int i = 0; i< tab.length ; i++){
    assertEquals(tab[i],-1);

}
```



PLAN


```
        Tableau.remplirTab(tab);
        for(int i = 0; i< tab.length ; i++){
            assertEquals(tab[i],1);
        }

        Tableau.supprOcc(tab,0);
        for(int i = 0; i< tab.length ; i++){
            assertEquals(tab[i],-1);

        }
```



En conclusion,

-Junit est un outil complexe mais très utile, propre au java.

-Pour d'autres langages, il existe aussi Jenkins ou Hudson.

-Il permet de développer en sachant si l'on créé de nouvelles possibilités ou fonctionnalités pour le programme que l'on veut développer tout en conservant les anciennes.